Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation

Laurent Bobelin, Arnaud Legrand, Márquez Alejandro González David, Pierre Navarro, Martin Quinson, Frédéric Suter, Christophe Thiery

▶ **To cite this version:**

Laurent Bobelin, Arnaud Legrand, Márquez Alejandro González David, Pierre Navarro, Martin Quinson, et al.. Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation. CCGrid 2012 – The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 2012, Ottawa, Canada. pp.19. hal-00650233v2

HAL Id: hal-00650233
https://inria.hal.science/hal-00650233v2

Submitted on 5 Jun 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation

Laurent Bobelin[1], Arnaud Legrand[1], David A. González Márquez[2]
Pierre Navarro[1], Martin Quinson[3], Frédéric Suter[4], Christophe Thiéry[3]
[1] *LIG, Grenoble University, Grenoble, France*
[2] *Departemento de Computacion, Universitad de Buneos Aires, Buenos Aires, Argentina*
[3] *LORIA, Nancy University, Nancy, France*
[4] *IN2P3 Computing Center, CNRS/IN2P3, Lyon-Villeurbanne,France*

***Abstract*—Conducting experiments in large-scale distributed systems is usually time-consuming and labor-intensive. Uncontrolled external load variation prevents to reproduce experiments and such systems are often not available to the purpose of research experiments, e.g., production or yet to deploy systems. Hence, many researchers in the area of distributed computing rely on simulation to perform their studies. However, the simulation of large-scale computing systems raises several scalability issues, in terms of speed and memory. Indeed, such systems now comprise millions of hosts interconnected through a complex network and run billions of processes. Most simulators thus trade accuracy for speed and rely on very simple and easy to implement models. However, the assumptions underlying these models are often questionable, especially when it comes to network modeling.**

In this paper, we show that, despite a widespread belief in the community, achieving high scalability does not necessarily require to resort to overly simple models and ignore important phenomena. We show that relying on a modular and hierarchical platform representation, while taking advantage of regularity when possible, allows us to model systems such as data and computing centers, peer-to-peer networks, grids, or clouds in a scalable way. This approach has been integrated into the open-source SIMGRID simulation toolkit. We show that our solution allows us to model such systems much more accurately than other state-of-the-art simulators without trading for simulation speed. SIMGRID is even sometimes orders of magnitude faster.

I. INTRODUCTION

The current evolution of computing and data management platforms to ever growing and more distributed infrastructures, such as Data Centers, Computing Grids, Clouds, Desktop Grids, or Peer-to-Peer (P2P) systems, raises new challenges in Computer Science. The unprecedented scale of these systems makes it difficult and costly to test new protocols or algorithms on production infrastructures. Indeed, such systems are inherently unstable, which hinders experiments reproducibility. Furthermore, real systems (e.g., a P2P network or a production computing center) are generally not available to the purpose of research experiments. Last, researchers may be interested in studying systems that do not exist yet for example when performing preliminary study or investigating the efficiency of solutions under workloads different from the ones available at hand. Hence, simulation has been extensively used to study such Large-Scale Distributed Computing (LSDC) systems.

Yet, simulating such LSDC systems raises scalability issues, both in terms of speed and memory. Indeed, such systems now comprise millions of hosts interconnected through a complex network and run billions of processes. The faithful representation of the interconnection network is generally difficult. A few simulators rely on a detailed model of the network (e.g., packet-level simulators that are heavily used in the network community) but their use in the context of distributed computing raises two difficult issues. First, such complex models are much harder to instantiate than simple ones (detailed topology, network characteristics of the whole platform, injection of realistic external background load at the packet-level, ...). Second, packet-level simulation is so heavy that it becomes completely unusable beyond a few thousands hosts. Hence, most simulators from these domains address scalability issues by relying on simplistic models that can be implemented efficiently although such simplifications are sometimes very debatable.

For example, P2P simulators generally rely on delay-based models and forget about the underlying physical topology. Using this approach, tools such as PeerSim [1] can simulate platforms that scale up to millions of nodes. Yet, such models do not account for network contention, whereas most peers generally sit behind asymmetric DSL lines with very limited bandwidth. Although this kind of assumption may not be harmful when studying simple overlays and investigating the efficiency of look-up operations in a Distributed Hash Table (DHT), the use of such simulators for streaming operations and file sharing protocols is much more controversial.

This need for a scalable but accurate network representation is not specific to P2P studies. Many simulators in the High Performance Computing (HPC) community assume that bandwidth has been over-provisioned and that contention can be ignored. Although this assumption may hold true for supercomputers, it may not for commodity clusters or future exa-scale platforms where energy consumption is at stake, which precludes resource over-provisioning.

The common belief in the distributed computing community is thus that, when scale is at stake, simplistic network models have to be used. In this article, we propose an efficient platform representation that is the final piece missing to the open-source SIMGRID simulation toolkit to dismantle this belief by showing that reasonably accurate network models can be used at large scale. We show that, while using a much more accurate and complex network model, our simulations are as fast or even orders of magnitude faster than state-of-the-

art simulators. Moreover, unlike other *ad hoc* simulators, the complete separation of application modeling from platform modeling enables SIMGRID based simulations to be evaluated seamlessly in a wide range of settings and to easily adjust the level of details to the scenario under investigation.

Section II details the main concerns related to network representation and modeling in a simulation context. Section III presents related work while Section IV exposes our proposal which is evaluated in Section V. We summarize our contributions and give some future work directions in Section VI.

II. Main Concerns about Network Representation and Modeling in Distributed System Simulation

A. Community Specific Requirements

Generally speaking, simulation needs to account for key characteristics of the system under study, that heavily depend on the investigated scenario. Having a general discussion about network modeling without specifying the context of use would be meaningless. Researchers that study the effectiveness of a modification on a network protocol clearly require accurate packet-level simulations whereas a researcher interested in the scalability of a grid middleware involving very large data transfers over long periods of time would only require a coarse grain modeling of the network. Hence, we start by listing some very commonly investigated scenarios in distributed systems to illustrate the kind of phenomenon that one may wish to be accounted for.

P2P DHT simulation generally needs to account for the geographic diversity of peers, communication jitter, and peer churn. Most DHTs rely on the exchange of small messages and contention can thus be somehow ignored. Nevertheless the amount of exchanged messages is generally a measure of interest. A large amount of messages may indicate that the contention-free hypothesis is broken and that conclusions should be disregarded.

P2P streaming simulation is more complex as it involves a higher network traffic. The key characteristics to account for are network proximity, connection asymmetry (since peers act as forwarders) and interference between concurrent connections on the borders of the network. In such settings, a detailed modeling of the topology is not important. Only the edge of the network has to be accurately modeled, as it generally constitutes the main bottleneck.

Volunteer computing studies imply the simulation of clients and/or servers. The key characteristics to account for are the volunteer dynamic availability, properties (e.g., CPU speed, number of cores, disk space, ...), and reliability (regarding the correctness of the returned results). In some cases it may be important to model the characteristics of the connection of the peers to the Internet as well. Then the same requirements as for P2P streaming simulations apply.

High Performance Computing studies generally involve a complex communication workload made of sequences of very short communications, complex synchronization patterns, and structured concurrent transfers of large amount of data. A faithful simulation of such workload requires a precise modeling of the network characteristics and protocols but can build on the regularity and homogeneity of the underlying network topology.

Grid Computing studies often involve numerous large data transfers across wide area networks that require an accurate modeling of contention and to account for both platform heterogeneity and hierarchy.

Cloud computing studies on IaaS may require a mixture of the previous requirements. For instance, it may be important to precisely model what happens within data centers. To reflect a hierarchical organization, high-end compute nodes are distinguished from low-end compute nodes and storage nodes. When the infrastructure comprises several sites, the wide area network connection between sites has also to be accounted for. Then it is possible to study the different charging mechanisms involved when going from one site to another. While a precise modeling of client connectivity may not be required, it is important to consider their geographic diversity at least.

All these scenarios illustrate a wide variety of needs for network modeling and representation. It partially explains why most simulators are domain specific.

B. Scalability Issues

The lack of interdisciplinary simulation approaches can also be explained by scalability requirements. To allow for studies at the desired speed and scale, most simulators build on and take advantage of the specifics of the scenarios they are designed for. When scalability is at stake, two related issues come into play that both involve space and time considerations:

- **Platform description**: Most simulators take a description of the platform as input. Its size depends on the expressiveness of the chosen description format. When some regularity in terms of topology or characteristics can be exploited, this description can be reduced to a minimal size and a program can turn this *compact description* into an appropriate memory representation. For instance, describing a homogeneous cluster or a set of peers whose speed is uniformly distributed does not require to detail each single entity. Such an approach greatly reduces both *platform description size* and *parsing time*. However, researchers often need to investigate complex scenarios to confirm or infirm hypothesis. Then details of the platform could be specified and modified at will, thus breaking these regularity assumptions. For such simulation scenarios a *flat description* of all components is needed, which greatly increases platform description size and parsing time.

- **Memory representation**: In most cases, flat descriptions are trivially translated into a *flat representation* leading to a large *memory footprint*. Even compact descriptions often have to be expended in memory. For instance, a homogeneous cluster of N machines can be solely described by N and the characteristics of one machine, e.g., CPU speed, latency, gap and overhead of the network. However, a simulator will have to keep track of the activity of every single machine. The size

of the description would then be $\Theta(1)$[1] while its memory representation would be $\Omega(N)$. The same occurs when describing a set of peers by a statistical distribution. Such expansion heavily impacts the parsing time. In some cases, the regularity can be preserved leading to a *compact memory representation*. For instance, in P2P DHT simulations, assuming that N peers are interconnected in a general tree topology implies that the platform description and data structure sizes are both in $\Theta(N)$. This calls for a modular platform representation that allows researchers to express regularity when possible to help the simulator and adjust the level of details at will.

C. Network Communication Modeling

Research on networks has been relying on simulation for a long time, leading to standards and many popular open-source projects [2], [3], [4], [5]. The validity is these tools is ensured by a wide usage and the cross reproduction of experiments. They simulate the entire protocol stack, i.e., each packet becomes an event. Then the simulation accuracy comes from a very fine modeling of the entire system. Unfortunately such simulations are very slow, hence limiting the scalability of studies on LSDC systems. Moreover, very fine models are not always meaningful at a very large scale. They require an instantiation with parameters that are often not available, such as a complete description of the Internet. At such scale, these models may be unstable and, badly instantiated, are worse than simple models. Then simulations of LSDC systems heavily rely on simpler models classified as follows:

• **Delay-based.** Representing the communication time by a constant delay or a statistical distribution is often sufficient. Such models account for heterogeneity but not for network proximity. Then many models determine delays based on coordinate-based systems [6]. They are a good trade-off as they account for important characteristics with a $\Theta(N)$ memory footprint and a $O(1)$ delay computation time. Since coordinates may change over time and suffer from measurements issues, noise can be added to these coordinates. Similarly LogP-like models [7], [8], [9] have been heavily used in HPC and are extremely scalable since both description size and delay computation time are in $O(1)$. However, these very scalable models ignore network congestion and typically assume an extremely large bisection bandwidth.

• **Contention-based.** Modeling contention without hindering scalability requires an abstraction of the network topology that focuses on the most important parts. For instance, the bottleneck of a P2P streaming application will be mainly on the edge of the network. Contention in the core of the network may safely be ignored. Conversely, studying workload on data grids or collective communications requires a more precise model of the core of the network.

Simplistic packet-level simulation, e.g., store-and-forward or wormhole, of communications is blatantly inaccurate as it ignores many of the key characteristics of deployed protocols such as TCP. The obtained results are thus very questionable at best. Furthermore, such simulations are generally rather slow and not scalable since they involve a lot of simulation events for a single communication.

A sounder alternative is flow-level simulation in which the whole communication, or flow, is simulated as a single entity. The time needed to transfer a message of size S between hosts i and j is then given by:

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \tag{1}$$

where $L_{i,j}$ (resp. $B_{i,j}$) is the network latency (resp. bandwidth) on the route connecting i and j. Although determining $L_{i,j}$ may be easy, estimating the bandwidth $B_{i,j}$ is more difficult as it requires to account for interactions with every other flow. Then the simulation amounts to solving a bandwidth sharing problem, i.e., determining how much bandwidth is allocated to each flow.

Given the computed bandwidth allocation,which defines all data transfer rates, and the size of the data to be transmitted by each flow, one can determine which flow will complete first. Upon completion of a flow, or upon arrival of a new flow, the bandwidth allocation can be reevaluated. Usually, this reevaluation is memory-less and does not depend on past bandwidth allocations. This approach makes it possible to quickly step forward through (simulated) time, and thus is attractive for implementing scalable simulations of LDSC systems with potentially large amounts of communicated data.

It has been shown in [10] that such models can account for the main characteristics of TCP connections. The *slow start* phenomenon models the fact that the TCP protocol does not send at full speed immediately and constantly adjust the emission rate to observed packet loss. Such mechanism breaks the linearity assumption of Eq. (1). A clever instantiation leads to a very good approximation for large enough messages [10]. For small messages, very good results can be obtained with a piecewise linear model [11].

The *flow control* mechanism of TCP is known to prevent full bandwidth usage as flows may be limited by large latencies [12]. Yet, it can be easily captured in a flow-level model by bounding the share a flow can get by the maximum congestion window size divided by the round trip time (RTT).

TCP is also known to be RTT-unfair. When two flows contend for bandwidth on a bottleneck link, they are assigned bandwidths inversely proportional to their RTTs [13]. As demonstrated in [10], the RTT time needs to account both for latency along the path but also for the possibly low capacity of some links that may incur large delays. Such RTT values can then be used to decide how to share bandwidth amongst flows contending on a bottleneck link.

Finally, it is very common when sitting behind a DSL connection that upload traffic negatively impact download traffic, due to delayed acknowledgments. This phenomenon is known as *ACK compression* and also occurs on full duplex high speed Ethernet links directly connecting two machines. The observed poor link utilization may be explained by a *data pendulum* effect where data and ACK segments alternatively

[1]Or $\Theta(\log(N))$ if we count the bytes needed to describe N. In the remaining, we omit this $\log(N)$ factor in all our complexity estimation as it is relatively negligible for the platform sizes we consider.

fill only one of the link buffers [14]. A fluid model based on max-min sharing can easily model the impact of such cross traffic [15].

Recent and thorough validation studies in [10], [15] plead for carefully designed and instantiated fluid models as a very reasonable approximation for most LSDC systems. Although one may fear that such models are computationally expensive, bandwidth sharing can be computed in time linear with the number of connections and their interactions using sparse and efficient data structures [16].

D. Network Topology and Routing Representation

While an adequate representation should allow for an efficient extraction of all the necessary information to simulate communications, there exists some non-trivial trade-offs between *information retrieval* and representation size. The cost of representing network topology and routing can thus be decomposed in terms relative to execution time and data size. The execution time is made of the *parsing time* to build the data structure and the *lookup time* to retrieve a given route from this structure. Data size is reflected by both the size of the input, i.e., the *description* file provided by the user, and the *memory footprint* of the corresponding data structure.

Table I
Θ COMPLEXITY OF NETWORK ROUTING REPRESENTATIONS.

Representation	Parsing	Lookup	Input	Footprint
Flat	N^2	1	N^2	N^2
Dijsktra	$N+E$	$E+N\log N$	$N+E$	$E+N\log N$
Floyd	N^3	1	$N+E$	N^2
Clique	N^2	1	N	N^2
Star	N	1	1	N
Cloud	N	1	N	N

Let us consider a set of N nodes interconnected by a general graph with E edges. Table I summarizes the corresponding costs for most common network representations. For all of them, the lookup time is actually in Θ(route size) and we assume a static routing (although some of these representations support dynamic routing). In the **flat representation**, each route is completely defined by the set of equipments belonging to it. It can represent arbitrarily complex platforms (even something that would not be a graph) and routing. The main drawbacks of such a representation are its management cost and poor scalability [17]. The **Dijkstra graph representation** proposed in [18] only stores information on shortest paths and enables a better scalability. Shortest path routing is only slightly restrictive since most Internet protocols implement such a routing. Furthermore, this representation allows the simulator to recompute the routing for every communication and then to model dynamic routing. However, this memory scalability comes at the cost of a lookup time several orders of magnitude larger [18]. It can be reduced by adding caches, which are completely ineffective in scenarios that have poor locality or involve a very large number of entities. The **Floyd graph representation**, also proposed in [18], is another way to store information only on shortest paths with different time and space requirements. While this approach reduces the description size and has a very good lookup time, its parsing time and memory footprint are prohibitive. Finally, network graphs, such as clique, star graphs or "clouds" (where each peer is connected to the core of the network where contention is ignored), exhibit a regularity that can be exploited thanks to an *ad hoc* **local routing table** and a specific routing management.

III. RELATED WORK

Many tools exist in the literature to simulate LSDC systems [1], [3], [4], [19], [20], [21], [22], [23], [24], [25], [26]. Most comes from, and are limited to, a specific research community. Table II details how each tool addresses the concerns related to network modeling and representation.

Table II
SIMULATION TOOLS ANSWERS TO NETWORK CONCERNS.

	Community	Input	Memory	Network model	Routing
NS-2	Network	API	flat	packet-level	static
NS-3	Network	API	flat	packet-level	static
Omnet++	Network	text	hierarchical	packet-level	static,dynamic
PeerSim	P2P	API	cloud	delay-based	static (direct)
OverSim	P2P	API	cloud	delay-based	static (direct)
SimBA	VC	text	none	delay-based	none
LOGGOPSIM	HPC	text	cloud	delay-based	static (direct)
GridSim	Grid	API	flat	delay-based	none
OptorSim	Grid	Text file	flat	contention-based	short. path
GroudSim	Grid,Cloud	API	cloud	contention-based	static (direct)
CloudSim	Cloud	Brite, API	flat	delay-based	dynamic (short. path)
SIMGRID	LSDC	XML, API	flat	Fluid	static (indirect)

IV. PROPOSAL

A. Target Platform Characteristics

Most current distributed systems mainly rely on three types of possibly interconnected and mixed networks: System and Local Area Networks (SAN and LAN), National Research and Education Networks (NRENs), or privately and independently managed networks. Such networks are mostly interconnected in a hierarchical way.

System and local area networks are generally organized in a very hierarchical way (e.g., with fat trees) or with large switches. Most of the time, they locally manage routing based on shortest path algorithms that differ by distance estimation, i.e., distance vectors (delay or throughput), link states, or hybrid, and how they converge upon failures. However, failures are rare on local networks and the routing behavior is thus most of the time equivalent to shortest path determination (in number of hops).

Local networks use gateways, also named point of presence or network access point, to access to Internet Service Providers (ISPs) or NRENs, which are, in turn, connected to higher level networks at the scale of a continent. These high level networks are named backbone or tier-1 networks in the Internet terminology. They use their own routing policy, also based on shortest path. The Border Gateway Protocol (BGP) is the most deployed algorithm on the Internet.

From a performance point of view, higher hierarchy subnetworks may use traffic aggregation and dynamic routing to perform load balancing. However studies have shown that no change may occur for 80% of the paths in a 24 hour

period [27]. Moreover, such changes may especially affect load balancing on backbone links, that are usually not communication bottlenecks. Then they can be ignored without any significant impact on simulation accuracy.

As a conclusion, large scale distributed systems are organized by a hierarchical aggregation of networks. Most aggregated networks are named Autonomous Systems (AS), as they behave independently from each other and may have very different structures. Each AS is connected with lower or higher level ASes by a set of entry points. This hierarchy is often bypassed by direct connections between same level ASes. Within an AS, a routing policy is applied, most of the time based on shortest path algorithms, e.g., OSPF or RIP.

B. Description Guidelines

Based on the aforementioned characteristics, we define a set of guidelines to build a satisfying network representation. First, the representation has to provide the user an easy way to directly describe the structure and entities he/she observes. Second, expressing hierarchy and composition of entities should be possible as it can reduce the simulation time. This expressiveness has to match community requirements from fine, e.g., router backplane, to coarse grain, e.g., cloud networks in P2P systems. Third, it should take advantage of the regular structures that exist in the network, such as cliques or stars, for sake of description simplicity. Fourth, a user should be able to specify that any AS can instantiate its own routing protocol. Finally, the path lookup and routing computation times should depend as less as possible on the size of the network to ensure a good scalability.

C. Proposal Representation Overview

None of the network routing representations presented in Section II-D exploits the hierarchy and the regularity of the platform. Each of them has its own pros and cons. The key to a scalable network representation is to provide users the ability to adapt the representation to their needs.

Our proposal is based on four semantic principles. First, we opt for *static routing* computation by default for scalability reasons. As stated in Section IV-A, it is sufficient for stable LSDC systems. Second, we use *fluid models* as they offer the best trade-off between accuracy and speed/memory footprint. Third, we take advantage of the *hierarchical* structure of current large scale network infrastructures, thanks to the concept of AS, be they local networks or conform to the classical Internet definition. Last, we allow users to specify platform representation within each AS, which allows us to take advantage of the regular structure of an AS when possible.

We propose stock implementations of well known platform routing representations, such as Dijkstra, Dijkstra with cache, Floyd, Flat, and rule-based. The first four were described in Section II-D. The rule-based model relies on regular expressions to exploit regular structures. Figure 1 shows an example of such a hierarchical network representation.

Each AS has one or more gateways, which are used to compute routes between ASes included in an AS of higher level. With this mechanism, the simulator can determine routes between hosts belonging to different ASes by looking for the first common ancestor in the hierarchy (see Figure 2). Routing is then solved recursively using the hierarchy. It allows us to represent hierarchical platforms in a very compact and effective way. However, as real platforms are not strictly hierarchical, we also define bypassing rules to manually declare alternate routes between ASes.

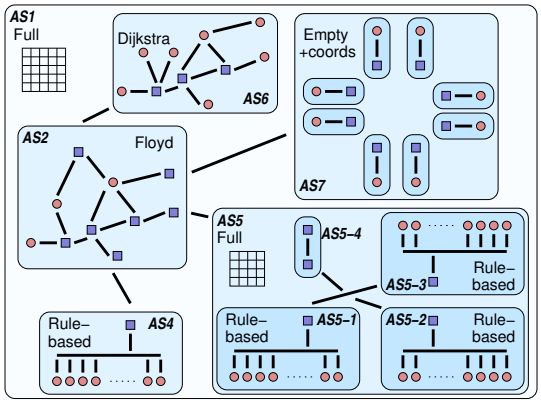

Figure 1. Illustration of hierarchical network representation. Circles represent processing units and squares represent network routers. Bold lines represent communication links. AS2 models the core of a national network interconnecting a small flat cluster (AS4) and a larger hierarchical cluster (AS5), a subset of a LAN (AS6), and a set of peers scattered around the world (AS7).

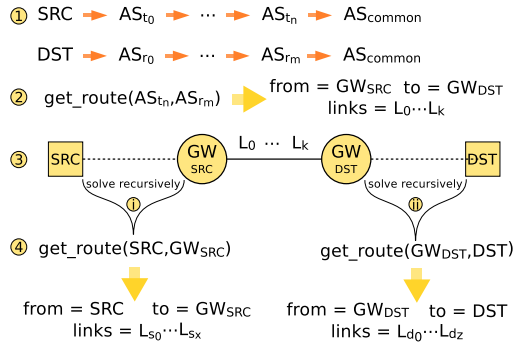

Figure 2. Main steps of the hierarchical routing mechanism.

In addition to these semantic principles, we also define some syntactical principles. We define the network representation as an XML file, as it is a common and easy to handle format. A user can then use standard XML editors and advance features such as auto-completion, validation, and well-formed checking. We also define a set of tags that simplify the definition of regular ASes, such as homogeneous compute clusters, or peers. The cluster tag creates a set of homogeneous hosts interconnected through a backbone and sharing a common gateway. The peer tag allows users to easily create P2P overlays by defining at the same time a host and a connection to the rest of the world (with different upload and download characteristics and network coordinates). With such a tag we benefit both from the compactness of coordinate-based models that account for delay heterogeneity and correlation, and from the accuracy of fluid models for contention. As such, the Vivaldi [6] and last-mile [28] models can be unified.

V. Evaluation

We claim that our proposal drastically reduces memory footprint without implying any prohibitive computational overhead. We show that this approach competes in terms of speed and memory usage with state-of-the-art simulators from various domains while relying on much more accurate models that are generally considered as prohibitive. For each domain, we first define a classical simulation scenario. Then, we simulate this scenario with SimGrid and the corresponding state-of-the-art simulator to evaluate their respective scalability.

Experiments were conducted a single core of a node with two AMD Opteron 6164 HE 12-core CPUs at 1.7 GHz with 48 GB of memory. SimGrid v3.7-beta, in which our proposal is integrated, was used. Source code, logs, platform files and analysis R scripts related to the experiments are freely available at http://simgrid-publis.gforge.inria.fr/xps/platform2011/.

A. Expressiveness

Table III
SIZE OF PLATFORM DESCRIPTION FILE IN VARIOUS SCENARIOS.

Community	Scenario	Size
P2P	http://pdos.csail.mit.edu/p2psim/kingdata/	209KB
P2P	http://www.cs.cornell.edu/People/egs/meridian/	301KB
P2P	http://www.eecs.harvard.edu/~syrah/nc/	28KB
Grid	10 sites, 40 clusters, 1500 nodes	22KB
HPC	4096 clusters of 64 nodes	27MB
Cloud	3 small data centers + Vivaldi	10KB

Table III exemplifies the flexibility and the compactness of our multi-purpose network representation proposal by giving the size of the XML input file in various scenarios.

B. Volunteer Computing

In [16], we compared the performance of SimGrid to that of SimBA. It is a discrete-event simulator that models hosts as finite-state automata with various parameters, i.e., compute, error, and timeout rates. Task failure or success depends on a uniform probability distribution while volatility is modeled using a Gaussian probability distribution. SimBA provides high scalability at price of simulation realism and is limited to the simulation of a single project. The SimGrid implementation simulates the whole infrastructure and accounts for complex scheduling decisions on every client, network contention, and churn. Here, SimGrid used the peer construct of Section IV-C and modeled host availabilities described by SETI@home traces from FTA [29]. About one third of the simulation time was spent parsing these traces.

The execution times measured with our simulator and those reported by Estrada *et al.* in [21] are contrasted as follows. Both timings were obtained on a 3.0 GHz Intel Pentium 4 with 1GB RAM (the one initially used in [21]). Estrada *et al.* simulated 15 days of computation with $7,810$ hosts working on the Predictor@Home project in 107 minutes. SimGrid simulates the same configuration in less than 4 minutes (for 10 runs the 95% confidence interval is $[208.13, 223.71]$ in seconds). Estrada *et al.* simulated 8 days of computation with 5,093 clients working on the CHARMM project in 44 minutes. SimGrid simulates the same configuration in less than 90 seconds (for 10 runs the 95% confidence interval is $[80.38, 80.87]$ in seconds).

While SimBA opts for many simplifications in its design and SimGrid for more sophisticated but costly options (e.g., use of host availability traces or simulation of processes that execute arbitrary code), SimGrid is still much faster.

C. P2P computing

Here we evaluate the impact of our proposal by comparing implementations of Chord simulations with various tools from the literature [1], [20] to an implementation on top of SimGrid. Chord was chosen because it is representative of a large body of algorithms from the P2P community and already implemented in most P2P simulators.

Figure 3 reports the simulation timing of the Chord scenario as a function of the node amount. It compares the results of OverSim when using its internal simple and scalable network underlay, OverSim using a detailed packet-level network model based on OMNet++ [19], and PeerSim to SimGrid using flow-based [10] or simplistic delay-based network models.

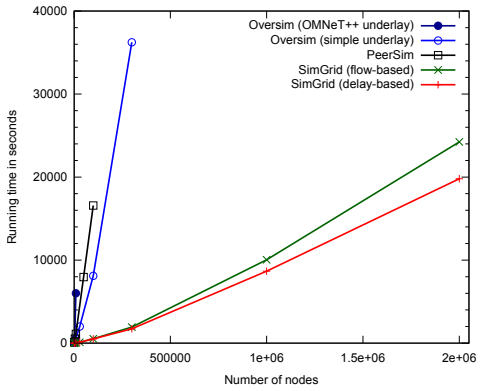

Figure 3. Running times of the Chord simulation with constant and precise network models on SimGrid, compared to Oversim with a simple underlay, OMNeT++ and PeerSim.

The largest scenario that we managed to run in less than 12 hours using OMNeT++ was $10,000$ nodes, in 1h40. With PeerSim, we managed to run $100,000$ nodes in 4h36 (but with a much lighter workload). With the simple underlay of OverSim, we managed to run $300,000$ nodes in 10h. With the precise flow-based model of SimGrid, we ran $2,000,000$ nodes in 6h43 while the simpler solely delay-based model of SimGrid ran the same experiment in 5h30. Simulating $300,000$ nodes with the flow-based model lasts only 32 minutes. The memory usage for simulating 2 million nodes in SimGrid was about 36 GB, that represents 18 kB per node, including 16 kB for the user code stack.

These results show that SimGrid is orders of magnitude more scalable than state-of-the-art P2P simulators. It is 15 times faster than the fastest ones, and simulates scenarios that are ten times larger. This trend remains when using the precise flow-based model, while simulation accuracy is improved.

D. High Performance Computing

In this section, we compare our proposal to the results published in [8]. LOGGOPSIM is a recent simulator specifically tailored for the study of MPI programs on large-scale HPC systems. It builds on the LogGPS model [9], which is a delay-based model accounting for middleware overhead and hardware gaps that prevent a perfect sustained usage of network cards. As explained in [8], the LogGPS model "ignores congestion in the network and assumes full effective bisection bandwidth", which is a valid hypothesis for several high-end computing platforms but not for cheaper Ethernet clusters or even torus-based networks as used in BlueGene or Cray machines. The model used in SIMGRID is an extension of the one proposed in [11] and accounts for such phenomenon as well as for network contention occurring in the core of the network. The importance of accounting for such phenomenon was demonstrated in [11] for MPI collective operations involving large amounts of data.

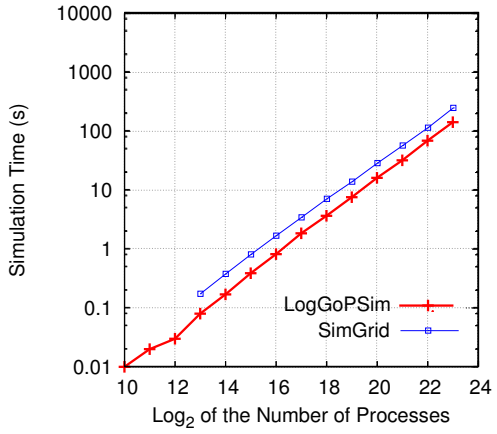

Figure 4. Performance comparison of LOGGOPSIM and SIMGRID when simulating a binomial broadcast.

LOGGOPSIM represents MPI executions as GOAL traces, i.e., DAGs of computations and communications. SIMGRID supports the evaluation DAG scheduling algorithms through the SIMDAG API since 2001. We aimed at comparing our proposal with LOGGOPSIM in the very same experimental setting used in section 4.1.2 of [8], i.e., the execution of a binomial broadcast on various the number of processes. Unfortunately, the input GOAL traces used in [8] were not available. Then we have compared our simulation results to the published one instead of reproducing the experiments with LOGGOPSIM. The evaluation of LOGGOPSIM was done on a 1.15 GHz Opteron workstation with 13 GB of memory. To perform a comparison as fair as possible we scaled down frequency of our processor down to 1GHz. Figure 4 shows the results. While using significantly more elaborate platform and communication models, and thus producing more meaningful results, SIMGRID is only roughly 75% slower than LOGGOPSIM. This demonstrates that scalability does not necessarily comes at the price of realism (e.g., ignoring contention on the interconnect). The genericity of SIMGRID data structures comes at the cost of a slight overhead since our memory usage for 2^{23} processors is 15GB, which is slightly larger than what is achieved in [8]. Yet, we think this loss is reasonable and amply offset by the gain in flexibility.

E. Grid computing

In this section we compare to GRIDSIM [22] (version 5.2 released on Nov. 25, 2010), a tool used widely in the research community. The scenario we propose is a simple master worker setting in which the master distributes N fixed size jobs to P workers in a round-robin way. In GRIDSIM, we did not define any network topology, hence only the output and input baud rate are used to determine transfer speed. For SIMGRID, we use a model of the Grid 5000 platform [30] (10 sites, 40 clusters, 1500 nodes), modeling each cabinet of the clusters as well as the core of the wide area network interconnecting the different sites. The experiments were conducted using an Intel Xeon Quadcore 2.40GHz with 8GiB of RAM.

The number of tasks n_{task} to distribute was uniformly sampled in the $[1; 500,000]$ interval and the number of workers W was uniformly sampled in the $[100; 2,000]$. We tried larger range at first but we faced the same scalability issue as already mentioned in [17]: GRIDSIM cannot run for more than 10,000 hosts because of Java thread management issues. We performed 139 such experiments for GRIDSIM (1000 for SIMGRID as it was much faster) and measured the wallclock time T (in seconds) and the memory consumption M (in kB). These experiments prove that T_{GRIDSIM} is quadratic in n_{task} and linear in W. As expected, the size (input, output and amount of computation) of the tasks to distribute have no influence, hence the following model (the R^2 of the linear regression is 0.9871):

$$T_{\text{GRIDSIM}} \approx 5.599\,10^{-2}W + 1.405\,10^{-8}n_{task}^2$$

Surprisingly, the memory, is not a simple polynom in W and n_{task}. It seems to be piecewise linear in both W and n_{task} (with a very steep slope at first and then a less steep one). Furthermore there are a few outstanding points exhibiting particularly low or high memory usage. All this can be probably explained by the garbage collection of Java. Hence, we only report the behaviour for the situation where the number of tasks is larger than 200,000 and the slope is not steep, taking care of removing a few outliers (R^2=0.9972):

$$M_{\text{GRIDSIM}} \approx 2.457\,10^{6} + 226.6W + 3.11n_{task}$$

Conducting the same analysis for SIMGRID shows that it is much more stable and requires a much smaller time and memory footprint (R^2 equal to 0.9984 and 1 respectively):

$$T_{\text{SIMGRID}} \approx 1.021\,10^{-4}W + 2.588\,10^{-5}n_{task}$$
$$M_{\text{SIMGRID}} \approx 5188 + 79.9W$$

This means that 5.2Mb are required to represent the Grid 5000 platform and the internals of SIMGRID. We did not optimize the stack size of processes (unlike when simulating P2P scenarios), which is why we have a payload of 80K per worker. Last, since tasks are allocated and deallocated on the fly, there is no visible dependency on n_{task}.

This clearly indicate that SIMGRID (with a flow-based network model and a very detailed network topology) is several orders of magnitude faster and smaller in memory than GRIDSIM (with a delay-based model and no network topology). To illustrate the time and memory scalability, the previous regressions indicate that GRIDSIM requires more than an hour and 4.4 GB for $N = 500,000$ and $P = 2,000$ while SIMGRID requires less than 14 seconds and 165MB.

VI. CONCLUSION AND FUTURE WORKS

In this paper, we reviewed existing approaches to represent large scale distributed computing systems in a simulation context. We characterized the underlying problem of the network representation and proposed an original and efficient multi-purpose representation. Our experiments showed that our approach is far more scalable than what is done by state-of-the-art simulators from any of the targeted research communities. We integrated this network representation into the SIMGRID tool. The chosen hierarchical and adaptive description based on the concept of Autonomous Systems also allows us to combine different kinds of platforms, e.g., a computing grid and a P2P system, to broaden the range of simulation possibilities.

We also demonstrated that the widespread beliefs that scalable simulations necessarily imply to simplify the network models and avoid the use of threads are erroneous. Our implementation within SIMGRID does not trade accuracy and meaning for scalability and also allows its users to simulate complex applications.

As future work we aim at considering the specifics of emerging systems such as IaaS Clouds. We also plan to further reduce platform description size (and hence parsing time) and memory footprint by exploiting stochastic regularity when available and by improving the programmable description approach. One particularly promising approach would be to inject randomness directly into the platform description, to model host availability or peer characteristics for instance.

ACKNOWLEDGMENTS

This work is partially supported by ANR (*Agence National de Recherche*), project reference ANR 08 SEGI 022 (USS SIMGRID) and ANR 11 INFRA 13 (SONGS). Special thanks to INRIA, which supported David Marquez's internship.

REFERENCES

[1] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. of the 9th Int. Conference on Peer-to-Peer Computing*, 2009.

[2] J. Cowie, D. Nicol, and A. Ogielski, "Modeling the Global Internet," *Computing in Science and Engineering*, vol. 1, no. 1, 1999.

[3] S. Mccanne, S. Floyd, and K. Fall, "The Network Simulator (ns2)," Available at http://nsnam.isi.edu/nsnam.

[4] "The ns-3 Network Simulator," Available at http://www.nsnam.org.

[5] G. Riley, "Simulation of large scale networks II: large-scale network simulations with GTNetS," in *Proc. of the 35th Winter Simulation Conference*, 2003.

[6] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," in *ACM SIGCOMM*, 2004.

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," in *ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, 1993.

[8] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model," in *Proc. of the 2nd Workshop on Large-Scale System and Application Performance*, 2010.

[9] F. Ino, N. Fujimoto, and K. Hagihara, "LogGPS: a parallel computational model for synchronization analysis," in *Proc. of the 8th ACM SIGPLAN symposium on Principles and practices of parallel programming*, 2001.

[10] P. Velho and A. Legrand, "Accuracy Study and Improvement of Network Simulation in the SimGrid Framework," in *Proccedings of the 2nd Int. Conference on Simulation Tools and Techniques (SIMUTools)*, 2009.

[11] P.-N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson, "Single Node On-Line Simulation of MPI Applications with SMPI," in *IPDPS*, 2011.

[12] M. Jain, R. Prasad, and C. Dovrolis, "The TCP Bandwidth-Delay Product Revisited: Network Buffering, Cross Traffic, and Socket Buffer Auto-sizing," Georgia Institute of Technology, Tech. Rep. GIT-CERCS-03-02, 2003.

[13] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi, and M. Roccetti, "TCP-Libra: exploring RTT fairness for TCP," UCLA Computer Science Department, Tech. Rep. 050037, 2005.

[14] M. Heusse, S. A. Merritt, T. Brown, and A. Duda, "Two-way TCP Connections: Old Problem, New Insight," *ACM CCR*, vol. 41, no. 2, 2011.

[15] P. Velho, L. Schnorr, H. Casanova, and A. Legrand, "Flow-level Network Models: Have we Reached the Limits?" INRIA, Research report RR-7821, 2011. [Online]. Available: http://hal.inria.fr/hal-00646896/en/

[16] B. Donassolo, H. Casanova, A. Legrand, and P. Velho, "Fast and Scalable Simulation of Volunteer Computing Systems Using SimGrid," in *Workshop on Large-Scale System and Application Performance*, 2010.

[17] W. Depoorter, N. De Moor, K. Vanmechelen, and J. Broeckhove, "Scalability of Grid Simulators : An Evaluation," in *Proc. of the 14th EuroPar Conference*, ser. LNCS, no. 5168. Springer, 2008.

[18] S. De Munck, K. Vanmechelen, and J. Broeckhove, "Improving The Scalability of SimGrid Using Dynamic Routing," in *Proc. of the 9th Int. Conference on Computational Science (ICCS)*, 2009.

[19] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proc. of the 1st Int Conf. on Simulation Tools and Techniques for Communications, Networks and Systems*, 2008.

[20] I. Baumgart, B. Heep, and S. Krause, "OverSim: A Scalable and Flexible Overlay Framework for Simulation and Real Network Applications," in *Proc. of the 9th Int. Conference on Peer-to-Peer Computing*, 2009.

[21] M. Taufer, A. Kerstens, T. Estrada, D. Flores, and P. Teller, "SimBA: A Discrete Event Simulator for Performance Prediction of Volunteer Computing Projects," in *Proc. of the 21st Int. Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2007.

[22] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *CCPE*, vol. 14, no. 13-15, 2002.

[23] W. Bell, D. Cameron, P. Millar, L. Capozza, K. Stockinger, and F. Zini, "OptorSim: A Grid Simulator for Studying Dynamic Data Replication Strategies," *IJHPCA*, vol. 17, no. 4, 2003.

[24] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds," in *CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing*, ser. LNCS, vol. 6586. Springer, 2010.

[25] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, 2011.

[26] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a Generic Framework for Large-Scale Distributed Experiments," in *Proc. of the 10th Int. Conf. on Computer Modeling and Simulation (UKSim)*, 2008.

[27] K. Butler, P. McDaniel, and W. Aiello, "Optimizing BGP security by exploiting path stability," in *Proc. of the 13th ACM Conference on Computer and Communications Security*, 2006.

[28] O. Beaumont, L. Eyraud-Dubois, and Y.-J. Won, "Using the Last-mile Model as a Distributed Scheme for Available Bandwidth Prediction," in *EuroPar*, ser. LNCS, vol. 6852. Springer, 2011.

[29] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems," in *CCGrid*, 2010.

[30] "Grid'5000," http://www.grid5000.org/.